Product usage instructions

VC5 Series PLC Custom Variable Usage Instructions

Suzhou VEICHI Electric Technology Co. Ltd
2023/6

Catalog

1. Variables

1.1 Custom Variables

In the VC5 programming system, in addition to programming directly using direct addresses, such as X, Y, M, D, R and other components for programming, it is also possible to program in the form of "variables" without specific storage addresses to achieve the desired control logic, or the complete control process of the application object, which improves the convenience and readability of code writing.。

Supported custom variables

Type	Capacity	Data Type	Description
Pointer	4096 Point（32bit）	BOOL/ INT/ DINT/ REAL Arrays	Pointer variables； Power down is not saved；
BOOL	2MB（8 bit）	INT/ DINT/ REAL Variables， INT/ DINT/ REAL Arrays， INT/ DINT/ REAL Combination Structures	256KB power down save， Other power down is not saved；
INT			
DINT			
REAL			

1.2 Define Variables

VC5 supports custom variables, and users can program directly with variable names in their programs by defining global variables and variable tables.
The following rules need to be followed when defining variable names:

1．Maximum length of 64 bytes；

2. Can only be composed of "_, letters, numbers, Chinese characters" and cannot start with "_, numbers"；

3. Cannot be renamed with "soft component forms, constants, standard data types, instructions"；

4. Cannot be "ARRAY, TRUE, FALSE, ON, OFF, NULL" and other keywords。

Variable data type

Variable definitions support structures and arrays, and variable data types are supported as follows:

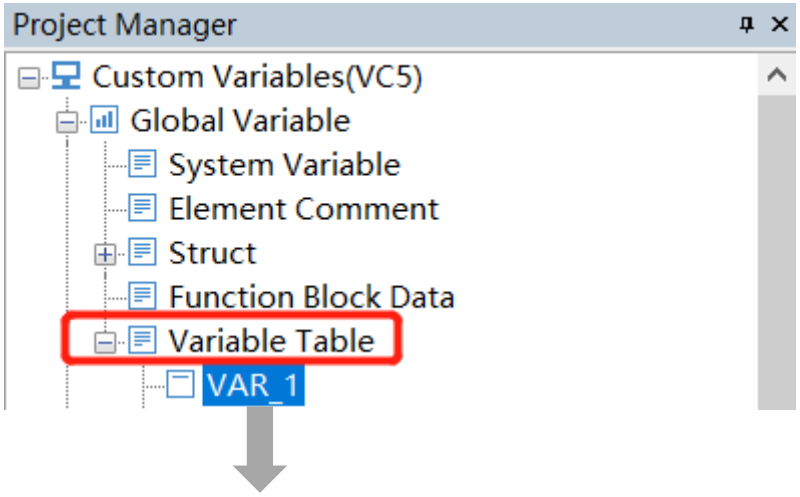
Variable data type

Data Type	Description
BOOL	Boolean type
INT	Single-word integer type
DINT	Double-word integer type
REAL	Real Number Type

Define Variables

The "Global Variables" in the Project Management section of the Auto Studio programming software is used for variable

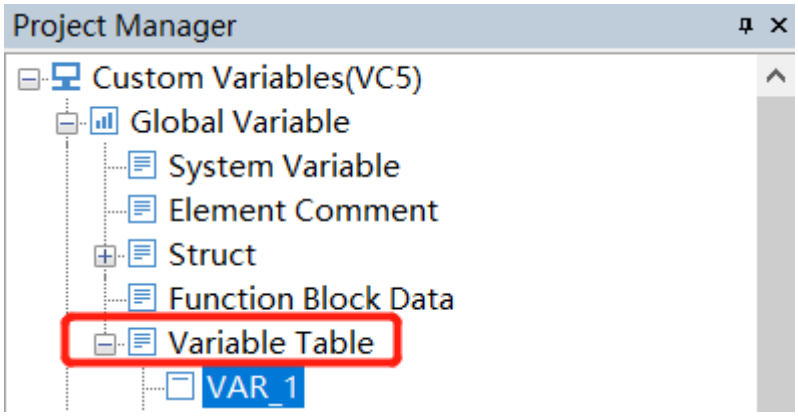
management, which allows you to add, delete and edit variables.。



MAIN VAR_1 x

Index	Variable Name	DataType	Initial Value	Data Hold	Element	Note
1	A	BOOL	OFF	Not Hold		
2	B	BOOL[3]	...	Not Hold		
3	B1	INT[3]	...	Not Hold		
4	var_array	INT[3]	...	Not Hold		
5						

1. Add variable table and variables: right-click "Variable Table" and select "New Variable Table" to create a new variable table.。



2. Double-click the variable table to enter the variable editing interface

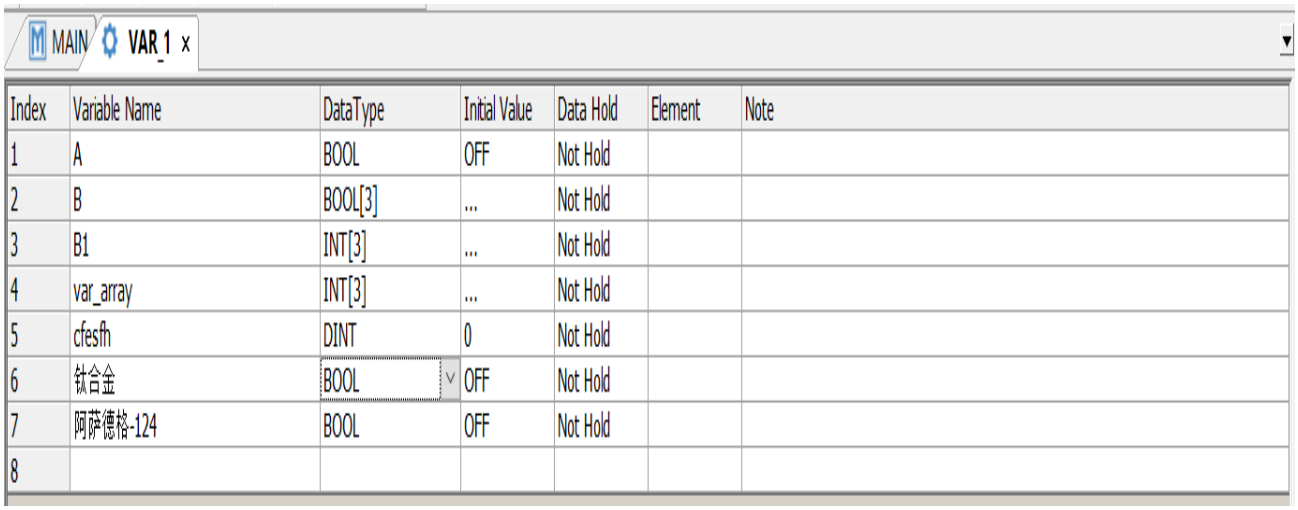
● In the variable table, right mouse click on the pop-up menu, you can insert or delete variables

● If you enter a custom variable name in the variable name column of the variable table, you can program directly with the variable name when programming.

● Data types can be selected from BOOL, INT, DINT, REAL, as well as arrays and structures (structures need to be defined in advance). When you select array as the data type, you can set the type and length of the array variable in the pop-up dialog box, and when you select a previously defined structure, you can define the structure variable.

● The initial value column can define initial values for variables, and arrays and structures can define initial values for each element individually

● Power-down hold can be selected for both hold and non-hold types, and the initial value setting is only valid for non-hold variables.

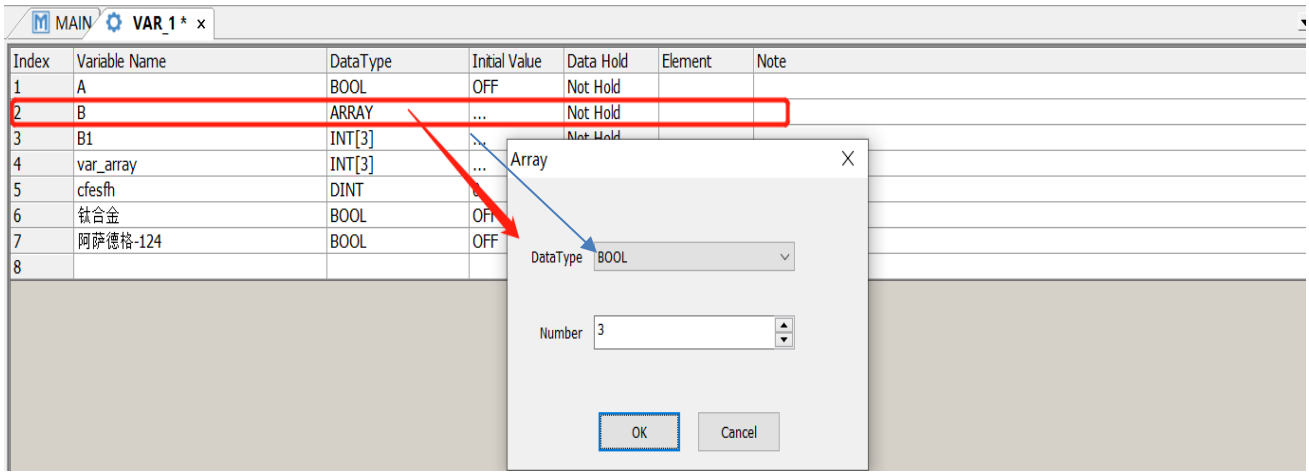


Index	Variable Name	DataType	Initial Value	Data Hold	Element	Note
1	A	BOOL	OFF	Not Hold		
2	B	BOOL[3]	...	Not Hold		
3	B1	INT[3]	...	Not Hold		
4	var_array	INT[3]	...	Not Hold		
5	cfesfh	DINT	0	Not Hold		
6	钛合金	BOOL	OFF	Not Hold		
7	阿萨德格-124	BOOL	OFF	Not Hold		
8						

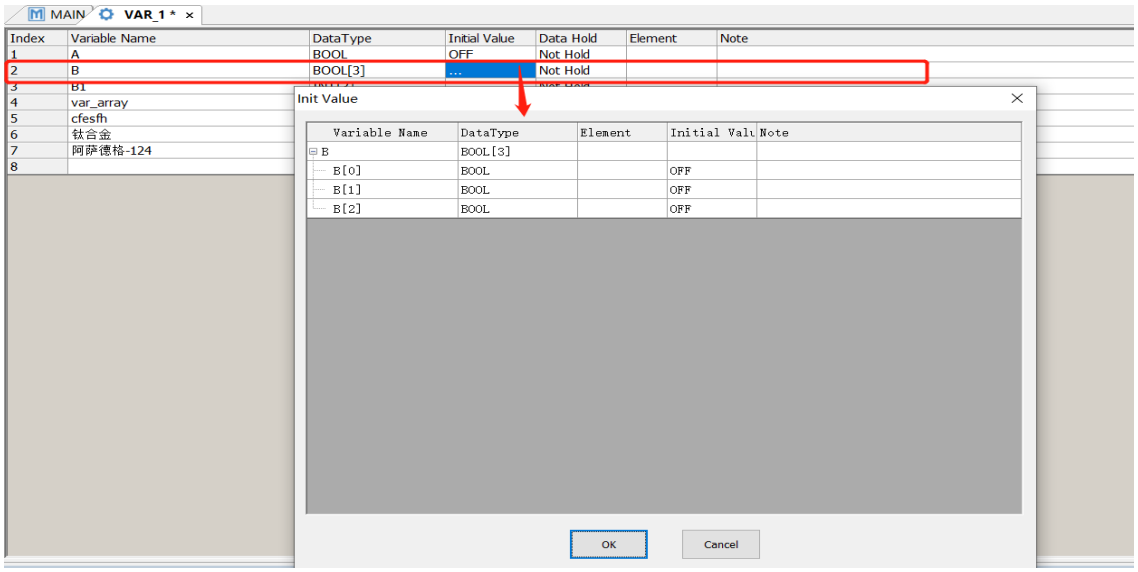
1.3 Defining Arrays

User programming can define arrays if the data type selected is ARRAY。

1. In the pop-up dialog box, select the type and length of the array variable, and click "OK" to define the array.。



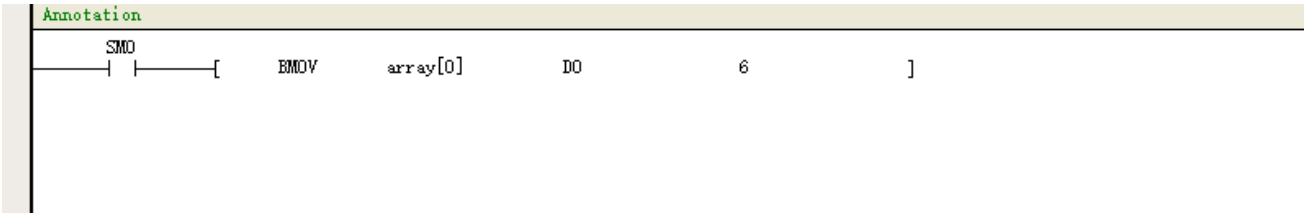
2. Click the initial value column of the array variable to enter the initial value setting interface of the array variable:



When an array is used in the command, the access starts from the element specified by the subscript of the array。

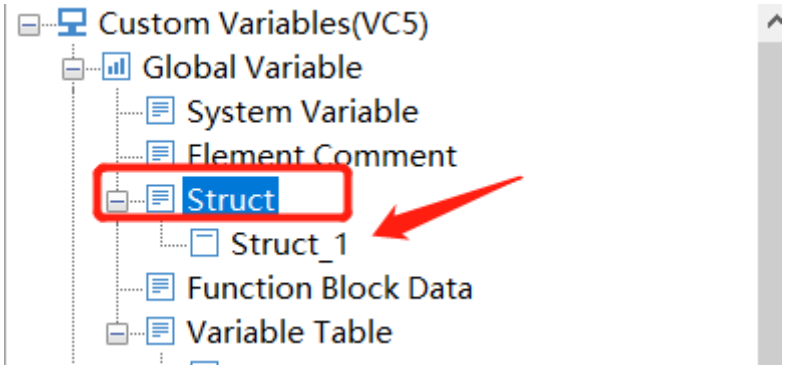
For example:

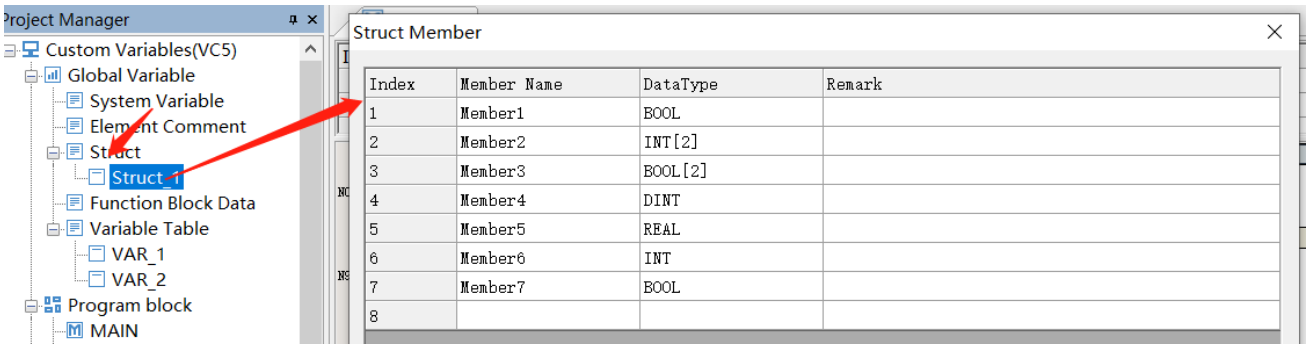
●Assign 6 elements of var_array [0] to var_array [5] to D0-D5



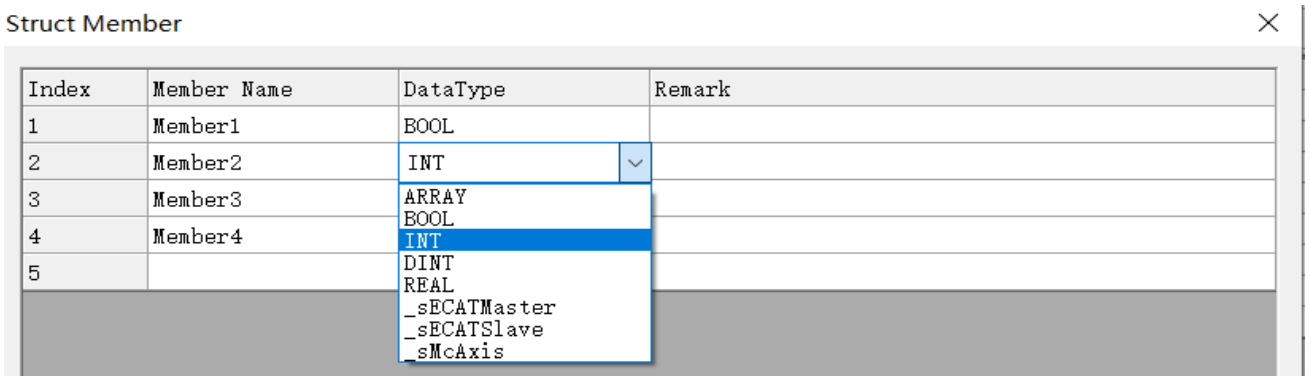
1.4 Defining Structs

If you need to define a structure variable in the variable definition, you need to define the data structure of the structure in advance. Right-click "Structure" under "Global Variables", select "New Structure", and enter the name of the structure to define the structure. When you define the variable in the variable table, you can select the structure type as the data type of the variable and define the variable as a structure variable.。

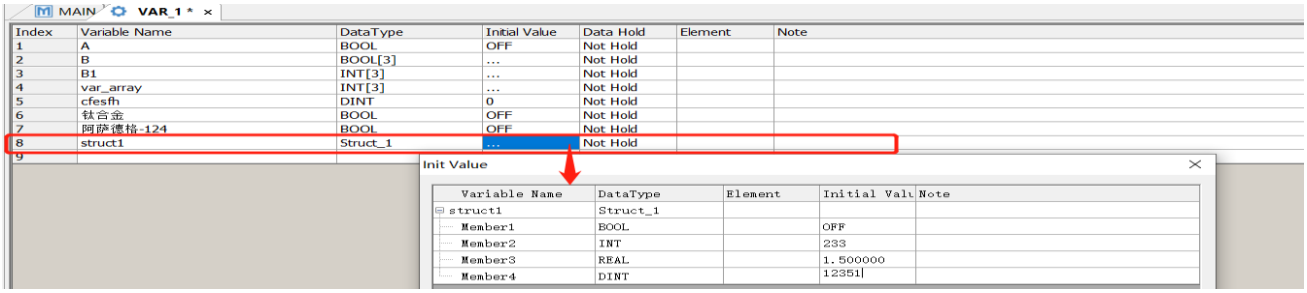




After creating structure and member variables, you can define structure variables by selecting structure in the data type of variable definition



Click the Initial value column of a structure variable to enter the initial value setting interface of the structure variable, and you can set the initial values of the structure variable members.。



1.5 How to use variables

Once the variables are defined, the variable names can be used directly in programming the program without the need to assign soft components。

● Direct variable programming operations.

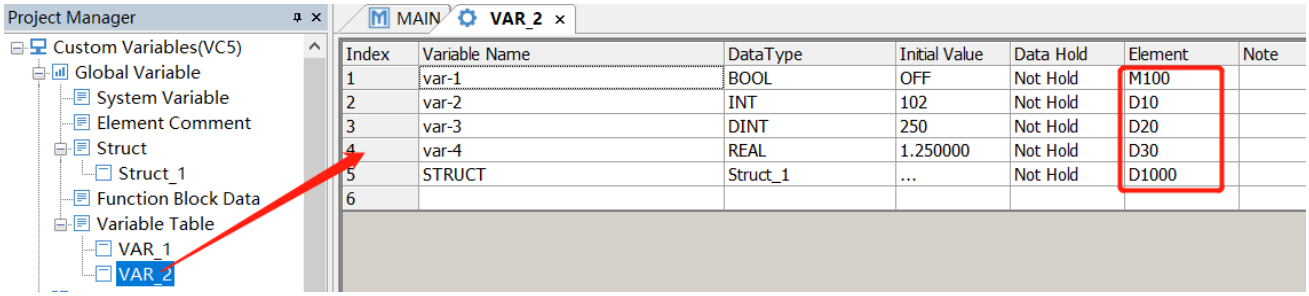
● When using array variables, the program uses "[number]" to represent the array elements, starting from 0.

● When using structure variables, program "structure variable name. Member Variable" to indicate a member of a structure.

2. Binding of variable addresses

2.1 Overview

The custom variables in VC5 support binding the addresses of soft components, and the addresses of custom variables are associated with the addresses of soft components after binding. To implement the function of custom variable binding software, just fill in the address field in the variable table with the address you need to associate, and then compile the project after inputting, the software will automatically generate the assigned address。

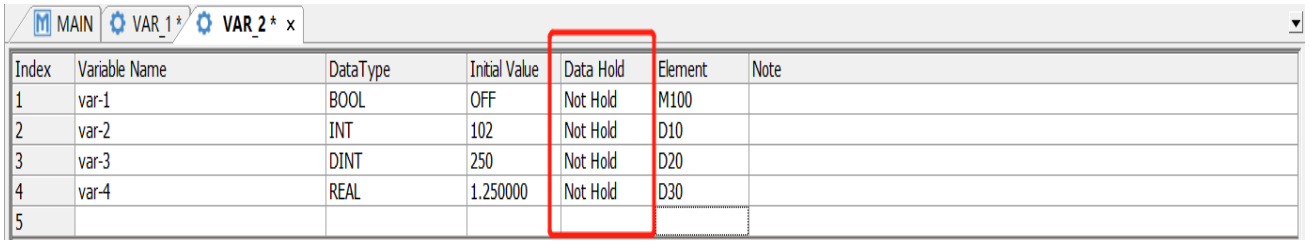


Index	Variable Name	DataType	Initial Value	Data Hold	Element	Note
1	var-1	BOOL	OFF	Not Hold	M100	
2	var-2	INT	102	Not Hold	D10	
3	var-3	DINT	250	Not Hold	D20	
4	var-4	REAL	1.250000	Not Hold	D30	
5	STRUCT	Struct_1	...	Not Hold	D1000	
6						

2.2 Variable Properties

After a custom variable is bound to a soft component, the power-down hold property will follow the bound soft component. As shown in the figure below, M100 is in the power-down hold area, so after Var_2 is bound to it, the power-down hold property becomes hold type accordingly, while D100 is in the non-power-down hold area, then Var_3 is bound to it as non-power-down hold type.

After binding the component, the power-down hold property will change automatically according to the situation, and the user does not need to set it.



Index	Variable Name	DataType	Initial Value	Data Hold	Element	Note
1	var-1	BOOL	OFF	Not Hold	M100	
2	var-2	INT	102	Not Hold	D10	
3	var-3	DINT	250	Not Hold	D20	
4	var-4	REAL	1.250000	Not Hold	D30	
5						

2.3 Array variable binding soft components

To bind soft components to an array variable, simply fill in the address field in the variable table with the address to be mapped.

1. Word variables occupy the corresponding number of subcomponents according to the variable type, one INT variable occupies one 16-bit component, and one REAL, DINT variable occupies two 16-bit components.
2. BOOL variables occupy the corresponding number of bit components.
3. Array variables can only bind soft components of corresponding types, i.e. word variables can only bind word components and bit variables can only bind bit components.

For example, if you define an array variable of BOOL type Array_0 with length 10 and specify to bind M0 components, it will occupy components M0-M9; if you define an array variable of INT type Array_1 with length 10 and specify to bind D0 components, it will occupy components D0-D9.。

2.4 Structure variable binding soft components

When binding soft components through structure variables, just fill in the address column in the variable table (note: the address can only be a word component, not a bit component), after filling in the address, click "Compile", the address of the structure member will be automatically generated by Auto Studio, the specific address assignment rules are as follows.

1. INT-type variables occupy one 16-bit component, and REAL and DINT-type variables occupy two 16-bit components.
2. Consecutive multiple BOOL types as a whole are aligned by 16 bits, that is, the first allocation address of 16-bit soft component bit0 for consecutive BOOL type members, and the allocation addresses of consecutive multiple BOOL type variables are incremented by 1 bit in turn; for non-consecutive BOOL types, each is handled independently by 16-bit alignment.
3. Arrays and structure variables are aligned by 16 bits as a whole.

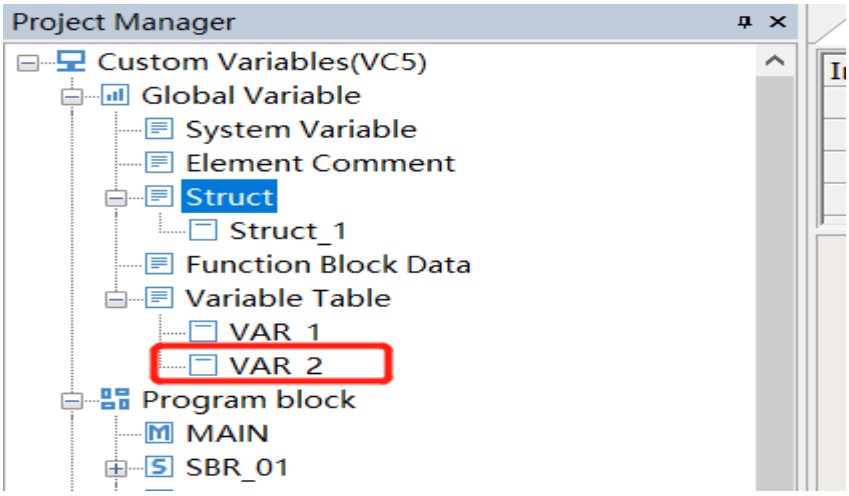
For example, define a variable Struct_1 of type Struct and specify the binding D1000 component

Struct Member

Index	Member Name	DataType	Remark
1	Member1	BOOL	
2	Member2	INT[2]	
3	Member3	BOOL[2]	
4	Member4	DINT	
5	Member5	REAL	
6	Member6	INT	
7	Member7	BOOL	
8			

- Struct t_1. member_1 // Type is BOOL, so bind D1000.0
- Struct_1. member_2 // type is an INT array, so bind D1001, D1002
- Struct_1. member_3 // Type is BOOL type array, so bind D1003.0, D1003.1
- Struct_1. member_4 // Type is DINT, so bind D1004
- Struct_1. member_5 //REAL，Therefore binding D1006
- Struct_1. member_6 // Type is INT, so bind D1008
- Struct_1. member_7 // Type is BOOL, so bind D1009.0

1. Structs are bound like other variables, just fill in the address field in the variable table with the address to be mapped.



MAIN | VAR_1 | VAR_2

Index	Variable Name	DataType	Initial Value	Data Hold	Element	Note
1	var-1	BOOL	OFF	Not Hold	M100	
2	var-2	INT	102	Not Hold	D10	
3	var-3	DINT	250	Not Hold	D20	
4	var-4	REAL	1.250000	Not Hold	D30	
5	STRUCT	Struct_1	...	Not Hold	D1000	
6						

After entering the address of the soft component, you need to compile the project, which will automatically generate the assigned address, then double-click the initial value column of the corresponding structure variable in the variable table, you can see the mapped address of each member of the structure and set the initial value of the variable。

Init Value

Variable Name	DataType	Element	Initial Valu	Note
STRUCT	Struct_1	D1000		
Member1	BOOL	D1000.0	OFF	
Member2	INT[2]			
Member2[0]	INT	D1001	0	
Member2[1]	INT	D1002	0	
Member3	BOOL[2]			
Member3[0]	BOOL	D1003.0	OFF	
Member3[1]	BOOL	D1003.1	OFF	
Member4	DINT	D1004	0	
Member5	REAL	D1006	0.000000	
Member6	INT	D1008	0	
Member7	BOOL	D1009.0	OFF	

3. Using variables as array subscripts

3.1 Rules of Use

The general rule for using variables as array subscripts: at most one variable in the entire variable composition is used as a subscript.

The format is defined as array[index] or stru[index].var, where array denotes an array or array of structures, index, var, i denotes a variable, stru denotes a structure, and so on。

Basic combination types

● array variables, which are used as variables for arrays and only support arrays of bit variables, arrays of word variables, arrays of double word variables, arrays of floating-point variables, etc., and do not support pointer variables;

● index variables, as variables with array subscripts, only support single word variables INT (16 bits) and double word variables DINT (32 bits), no soft components, no other variables such as bit variables, floating point variables, pointer variables, etc.; support a definite element of an array or a definite member of a structure as index variables, such as array[index[5]], array[stru.index], and does not support array elements with variable subscripts or array members of a structure as index variables, such as array[index[i]], array[stru[i].index];

Complex combination types

● Supports using elements of arrays (elements) as operands of instructions, i.e. index variables at the end, e.g. array[index], stru. array[index], stru1[3]. stru2. array[index], stru1.stru2. stru3. array[index] etc.

● Supports using members of the array of structures as operands of the instruction, i.e. the index variable is placed in the middle, e.g. stru[index]. Var, stru1[index]. stru2.var, stru1.stru2[5]. stru3[index]. array [3] etc.。

● The use of a single structure element in an array of structures as an operand of an instruction is not supported, i.e., the index variable is placed at the end, such as stru[index], stru1.stru2. stru3[index], stru1.stru2[2]. stru3.stru4[index], etc.

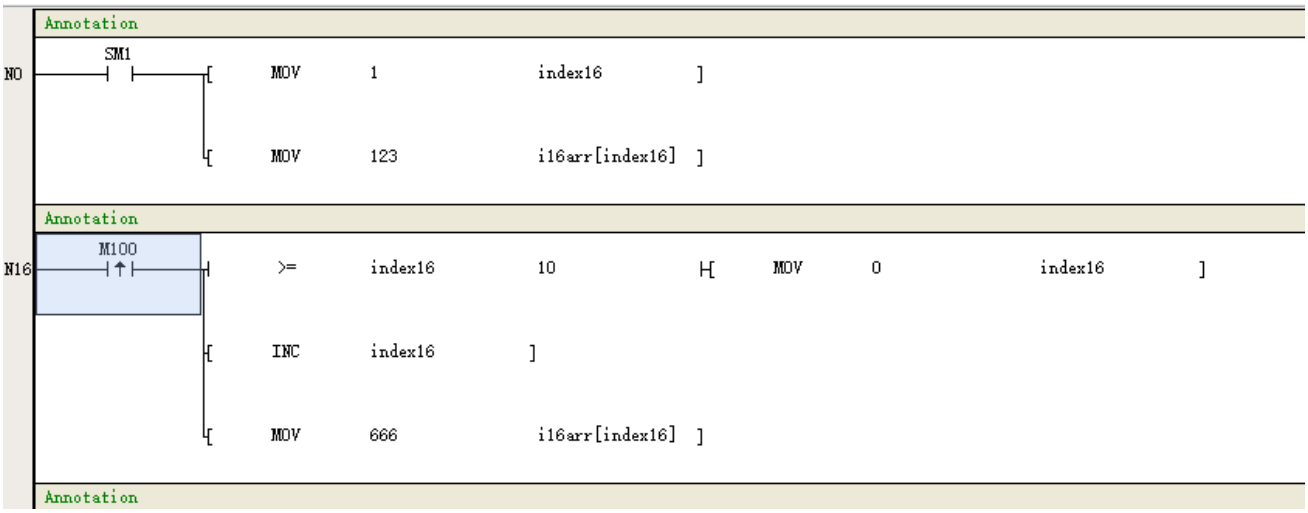
● Arrays of structures with double or multiple variables, such as stru[index1]. array[index2], are not supported.

● Two-dimensional or multi-dimensional arrays array[index1] [index2] are not supported.

3.2 Programming Examples

3.2.1 Example 1

To assign a value to an element of an array, the program is as follows:



Start assigning 123 to i16arr[1], after that, each time M100 is triggered, it will assign 666 to the array elements behind it in a smooth manner; after starting:

Output Window

	Element Name	data type	display format	current value	new value
1	i16arr	Array	Decimal		
2	i16arr[0]	INT	Decimal	0	
3	i16arr[1]	INT	Decimal	123	
4	i16arr[2]	INT	Decimal	0	
5	i16arr[3]	INT	Decimal	0	
6	i16arr[4]	INT	Decimal	0	
7	i16arr[5]	INT	Decimal	0	
8	i16arr[6]	INT	Decimal	0	
9	i16arr[7]	INT	Decimal	0	
10	i16arr[8]	INT	Decimal	0	
11	i16arr[9]	INT	Decimal	0	

After M100 triggers once:

Output Window

	Element Name	data type	display format	current value	new value	element remark
1	i16arr	Array	Decimal			
2	i16arr[0]	INT	Decimal	0		
3	i16arr[1]	INT	Decimal	123		
4	i16arr[2]	INT	Decimal	666		
5	i16arr[3]	INT	Decimal	0		
6	i16arr[4]	INT	Decimal	0		
7	i16arr[5]	INT	Decimal	0		
8	i16arr[6]	INT	Decimal	0		
9	i16arr[7]	INT	Decimal	0		
10	i16arr[8]	INT	Decimal	0		
11	i16arr[9]	INT	Decimal	0		

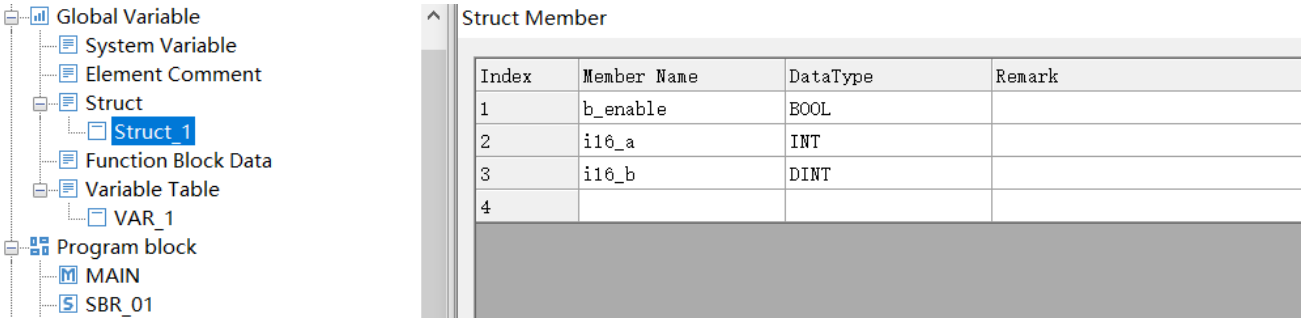
M100 after multiple triggers:

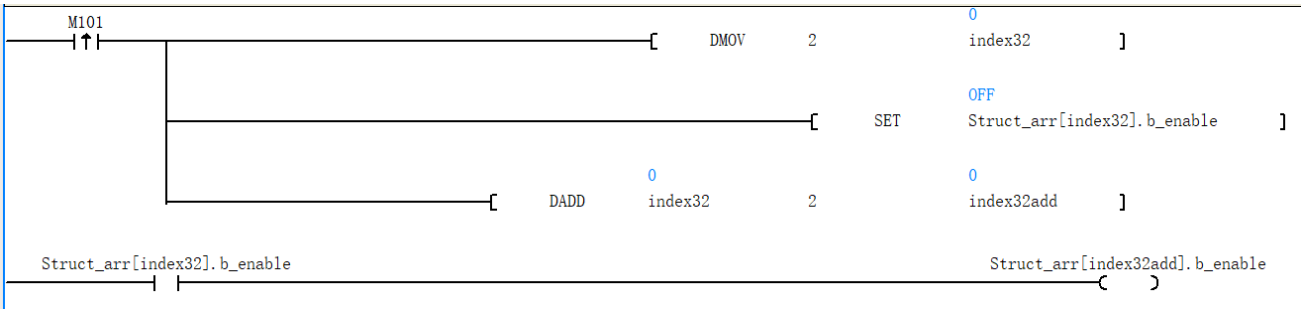
Output Window

	Element Name	data type	display format	current value	new value	element remark
1	i16arr	Array	Decimal			
2	i16arr[0]	INT	Decimal	0		
3	i16arr[1]	INT	Decimal	123		
4	i16arr[2]	INT	Decimal	666		
5	i16arr[3]	INT	Decimal	666		
6	i16arr[4]	INT	Decimal	666		
7	i16arr[5]	INT	Decimal	666		
8	i16arr[6]	INT	Decimal	666		
9	i16arr[7]	INT	Decimal	666		
10	i16arr[8]	INT	Decimal	666		
11	i16arr[9]	INT	Decimal	0		
12		INT	Decimal			

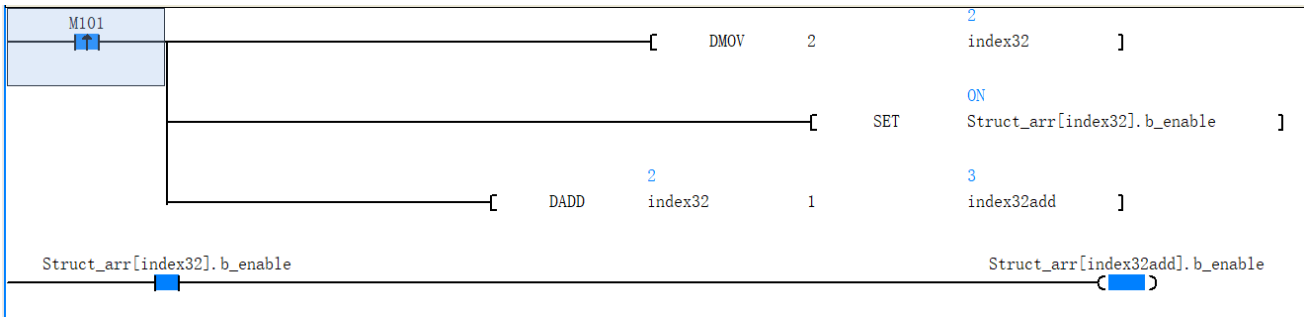
3.2.2 Example 2

Operate on a member variable of a structure array, structure definition:





After M101 is triggered, Struct_arr[2].b_enable is set, and according to its state, Struct_arr[3].b_enable is controlled:



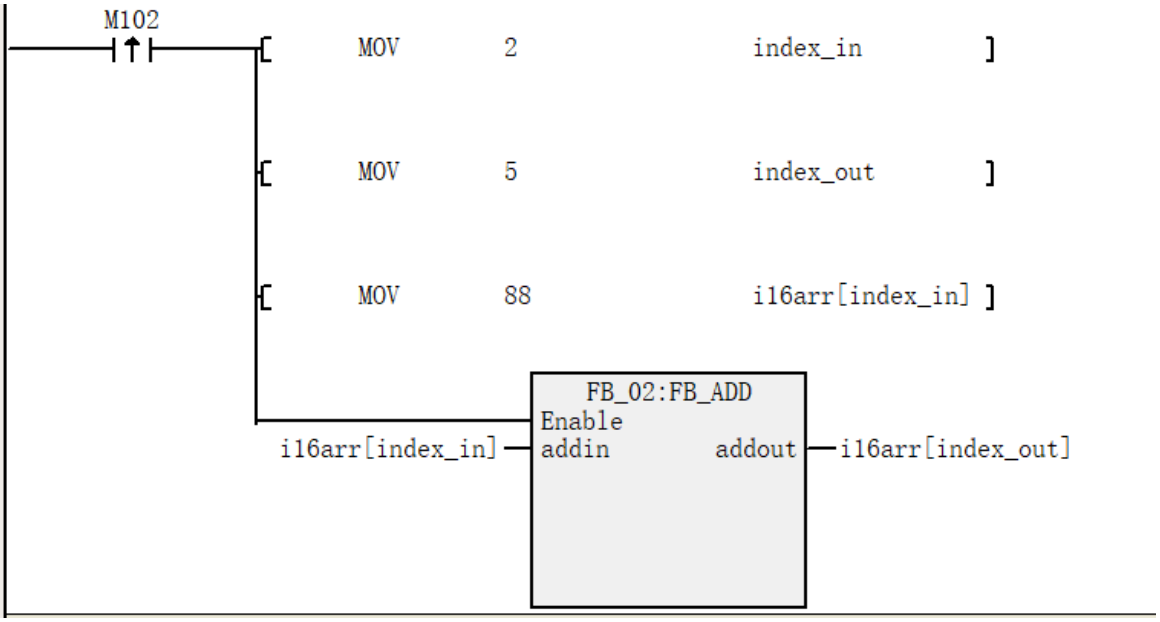
Output Window

	Element Name	data type	display format	current value	new value	element remark
13	struct_1[0]	Struct	Decimal			
14	b_enable	BOOL	Binary	OFF		
15	i16_a	INT	Decimal	0		
16	i16_b	DINT	Decimal	0		
17	struct_1[1]	Struct	Decimal			
18	b_enable	BOOL	Binary	OFF		
19	i16_a	INT	Decimal	0		
20	i16_b	DINT	Decimal	0		
21	struct_1[2]	Struct	Decimal			
22	b_enable	BOOL	Binary	ON		
23	i16_a	INT	Decimal	0		
24	i16_b	DINT	Decimal	0		
25	struct_1[3]	Struct	Decimal			
26	b_enable	BOOL	Binary	ON		
27	i16_a	INT	Decimal	0		
28	i16_b	DINT	Decimal	0		
29	struct_1[4]	Struct	Decimal			
30	b_enable	BOOL	Binary	OFF		
31	i16_a	INT	Decimal	0		
32	i16_b	DINT	Decimal	0		
33		INT	Decimal			

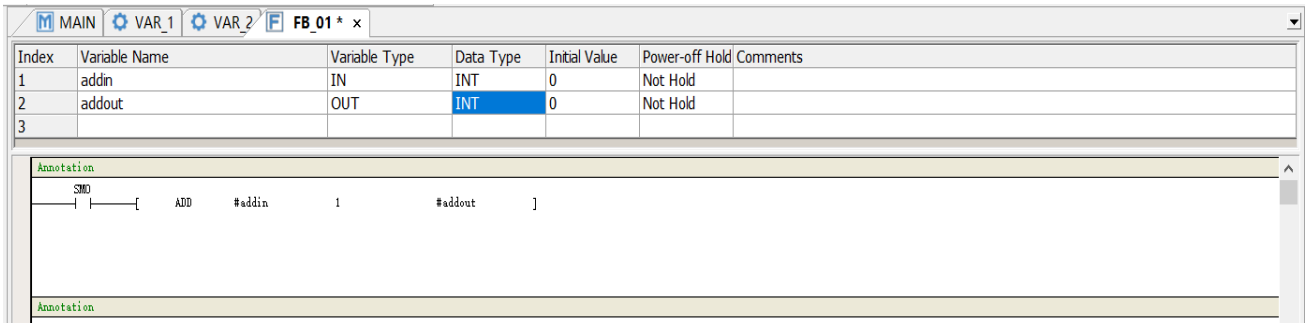
Compile | Communication | Convertion | Find | Monitoring

3.2.3 Example 3

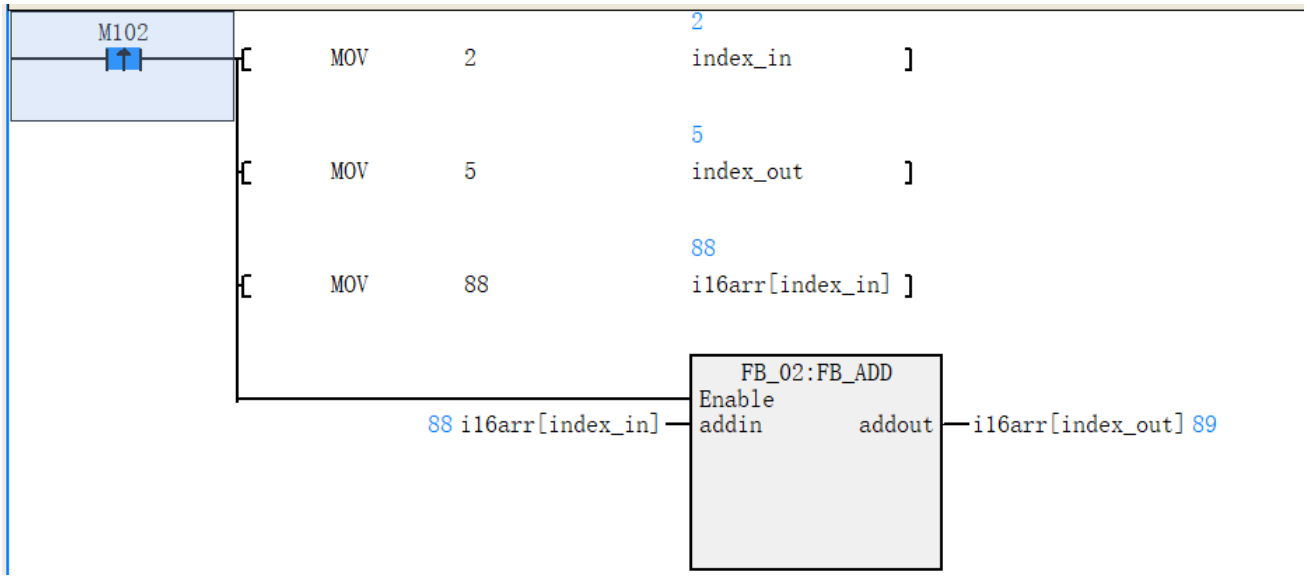
FB parameters use variables as array subscripts, and the program is as follows:



The FB program is as follows:



After M102 is triggered, the value 88 is assigned to i16arr [2], and after FB calculation, the value is assigned to i16arr [5]. The procedure of FB is to add 1 to the input parameter and assign it to the output。



Output Window

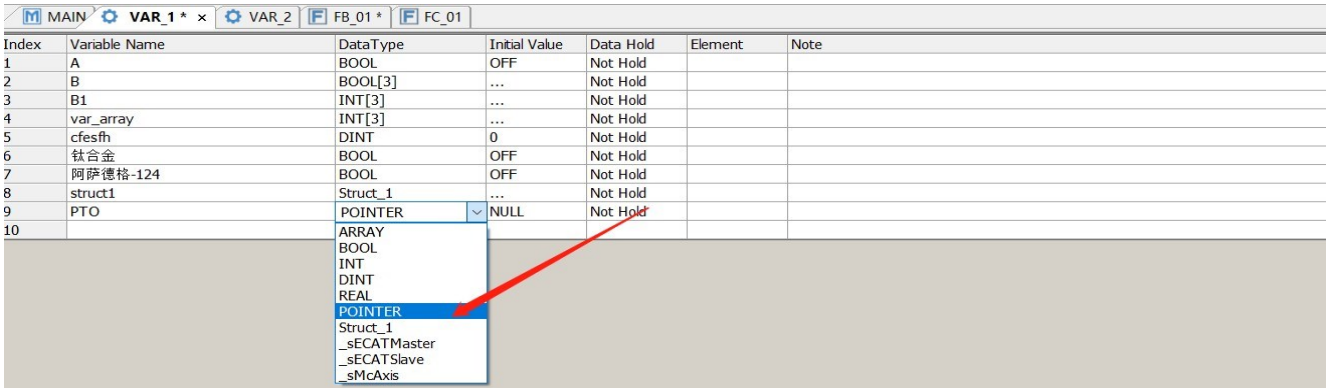
	Element Name	data type	display format	current value	new value	element remark
1	i16arr	Array	Decimal			
2	i16arr[0]	INT	Decimal	0		
3	i16arr[1]	INT	Decimal	0		
4	i16arr[2]	INT	Decimal	88		
5	i16arr[3]	INT	Decimal	0		
6	i16arr[4]	INT	Decimal	0		
7	i16arr[5]	INT	Decimal	89		
8	i16arr[6]	INT	Decimal	0		
9	i16arr[7]	INT	Decimal	0		
10	i16arr[8]	INT	Decimal	0		
11	i16arr[9]	INT	Decimal	0		

4. Pointer type variables

4.1 Definition of pointer type variables

A pointer variable can be used as a pointer to store the address of a soft component or an array variable, and can be used as indirect addressing or variable addressing when programming a program.

A pointer variable is defined when the variable name is defined in the variable table and "POINTER" is selected as the data type. The initial value of a pointer variable is NULL, i.e., a null pointer, and a pointer variable is not held when power is lost.



Pointer type variables can do address operations and indirect addressing operations. When using the pointer address operation instruction, it indicates the address operation of a pointer. The instructions that support pointer address operations are listed below. When using these instructions, the functions of fetching address, pointer address offset, and pointer address comparison are implemented.

Instructions for pointer address operations

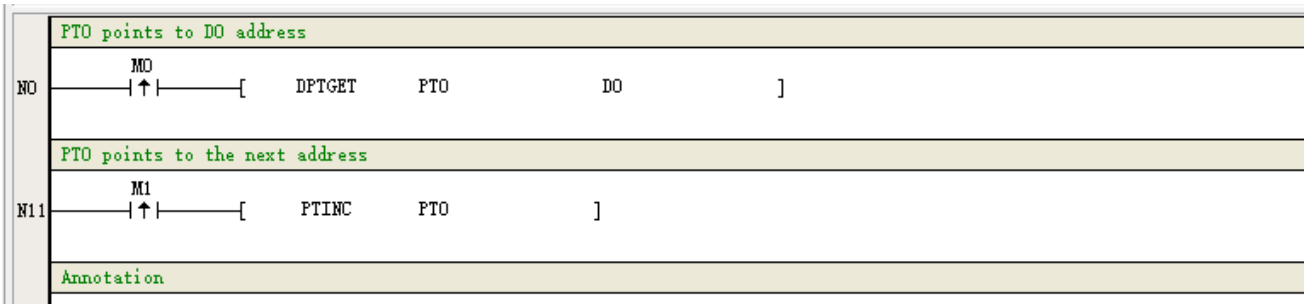
Instruction	Description
PTGET	Pointer variable assignment instructions
DPTGET	Pointer variable assignment instruction (double word)
RPTGET	Pointer variable assignment instruction (floating point)
PTINC	Pointer variable address increment 1 instruction
PTDEC	Pointer variable address minus 1 instruction
PTADD	Pointer variable addition instructions
PTSUB	Pointer variable subtraction instructions
PTMOV	Pointer variable address mutual assignment instruction
PTLD=	Pointer variable contact comparison equals instruction
PTLD<	Pointer variable contact comparison less than instruction
PTLD>	Pointer variable contact comparison is greater than the instruction
PTLD<>	Pointer variable contact comparison does not equal instruction
PTLD>=	Pointer variable contact comparison is greater than or equal to the instruction
PTLD<=	Pointer variable contact comparison less than or equal to instruction
PTAND=	Pointer variable and contact comparison equals instruction
PTAND<	Pointer variable and contact comparison less than instruction
PTAND>	Pointer variable with contact comparison larger than the instruction
PTAND<>	Comparison of pointer variables and contacts is not equivalent to the instruction
PTAND>=	Pointer variable and contact comparison is greater than or equal to the instruction
PTAND<=	Pointer variables are compared with contacts less than or equal to

	the instruction
PTOR=	Pointer variable or contact comparison equals instruction
PTOR<	Pointer variable or contact comparison less than instruction
PTOR>	Pointer variable or contact comparison is greater than the instruction
PTOR<>	Pointer variables or contact comparisons are not equivalent to instructions
PTOR>=	Pointer variable or contact comparison greater than or equal to instruction
PTOR<=	Pointer variable or contact comparison less than or equal to instruction

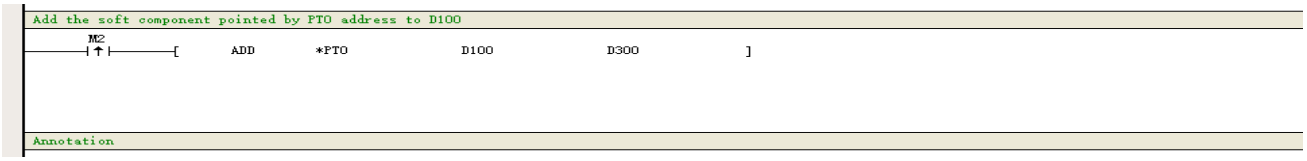
Except for the instructions in the table above that operate on pointer addresses, other instructions that use pointer type variables indicate indirect addressing operations on pointer type variables, i.e., operations on the values of soft components or array variables pointed to by pointer type variables. Indirect addressing of pointer variables is indicated by "*pointer type variable" in programming。

Example

● Address manipulation of pointer type variables



● Indirect addressing of pointer type variables



The programming software will automatically add "*" in front of variables of pointer types used in instructions other than those in the above table for pointer address operations.

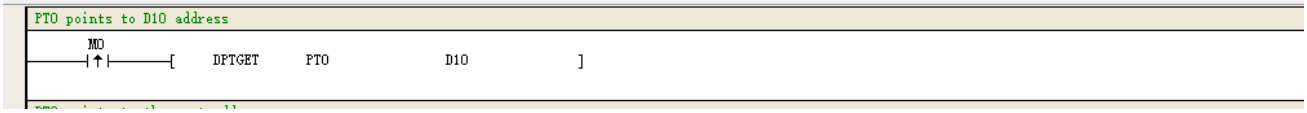
4.2 PT Pointer Address Operation

1. Getting the pointing address of a pointer variable

The pointer type variable pointing address can be obtained by the pointer variable assignment instruction (PTGET).

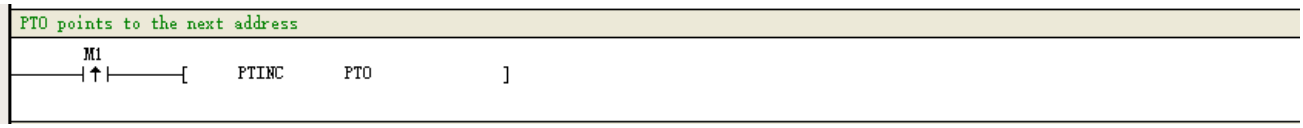
Example 1

When the instruction energy stream is valid, the pointer types variable PT0 points to D10, i.e. PT0 gets the address of D10 soft component.



Pointer type variables can point to bit components (X, Y, M, S), word components (D, R, W), and custom array variables. After a pointer type variable acquires a pointer address, you can add or subtract operations to the pointer address of the pointer type variable to indicate the component offset to which the pointer types variable points.

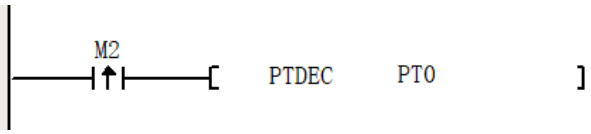
Example 2



If the instruction is valid, it will offset the soft component pointed by the pointer variable PT0 by one. If PT0 points to D10, after executing PTINC instruction, PT0 points to D11. After executing PTINC, the system will automatically adapt the offset by one according to the type of component or array variable pointed by the pointer variable.

PT0 Current Pointer	PT0 pointer after executing PTINC
D10	D11
M200	M201
Array [2]	Array [3]

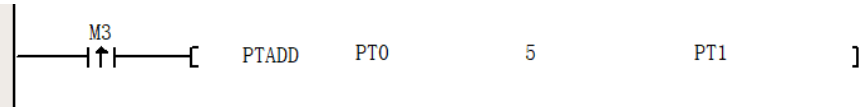
Example 3



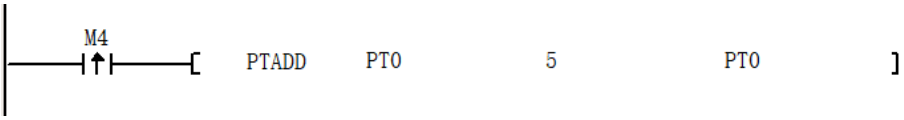
If the instruction is valid, the offset of the soft component pointed by the pointer variable PT0 will be reduced by one. For example, if PT0 points to D10, after executing PTINC instruction, PT0 points to D9. After executing PTDEC, the system will automatically adapt the offset minus one according to the type of component or array variable pointed by the pointer type variable.

PT0 Current Pointer	PT0 pointer after executing PTDEC
D10	D9
M200	M199
Array [2]	Array [1]

Example 4

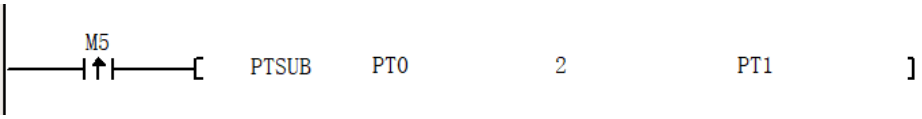


When the instruction is valid, the soft component offset pointed by the pointer address of PT0 is assigned to PT1 by 5, e.g. PT0 points to D10, after executing the example instruction, PT1 points to D15.



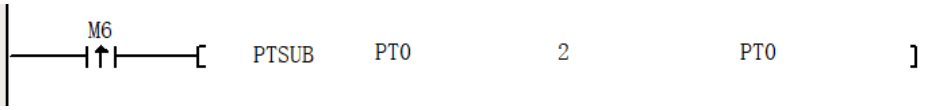
When the instruction is valid, the soft component offset of the PT0 pointer address will be assigned to PT0 by 5, e.g. PT0 originally points to D10, after the execution of the example instruction, PT0 points to D15.

Example 5



When the instruction is valid, the soft component offset of the pointer variable PT0 is assigned to PT1 minus two, e.g. PT0 points to D10, after the execution of the example instruction, PT1 points to D8.

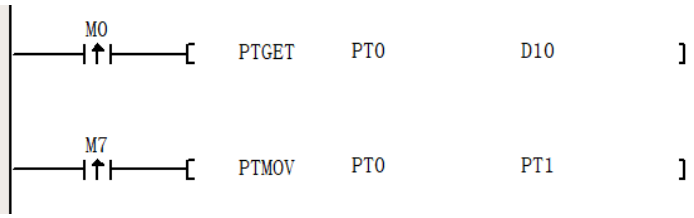
PT0 Current Pointer	After executing the command	
	PT1 Pointer	PT0 Pointer
D10	D8	D10



When the instruction energy stream is valid, the soft component offset of the PT0 pointer address minus 2 is assigned to PT0, e.g. PT0 originally points to D10, after executing the example instruction, PT0 points to D8.

Example 6

Pointer assignment to each other is the address backup of pointer variables, i.e., pointing to the same address;

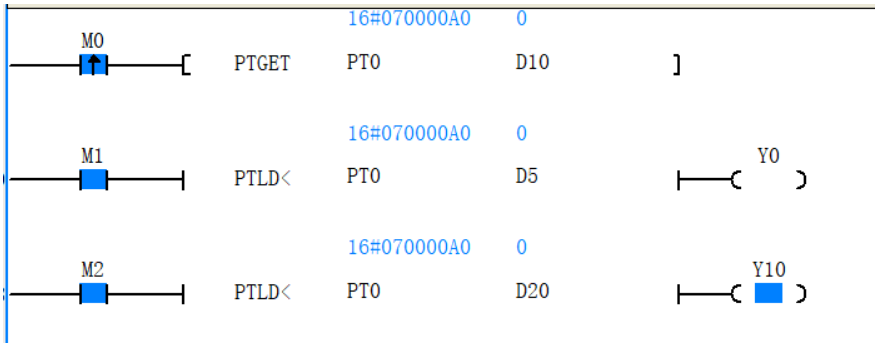


When the instruction energy stream is valid, the PT1 pointer points to the position of the PT0 pointer, i.e., D10

All of the above examples require the PTGET instruction to obtain the pointer address first; when PT points to an array variable, pay attention to the out-of-bounds check when executing the address operation instruction

Example 7

The contact comparison of a pointer is a comparison of the address of the pointer pointing to a soft component with the address of the object of comparison

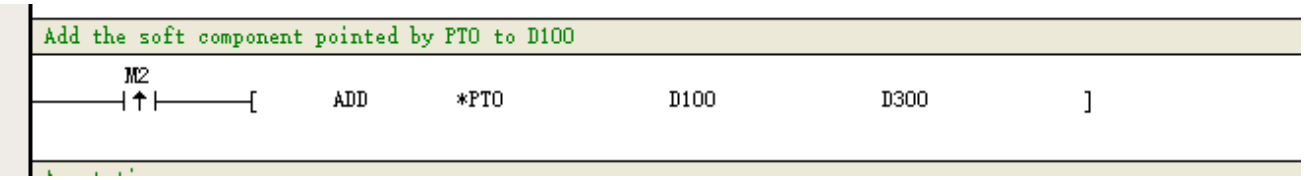


The PT< instruction checks the address of the pointer type variable PT0 to determine whether it is greater than the D address, e.g., if the PT0 pointer points to D10, a comparison with D5 will output Y0 as OFF, and a comparison with D20 will output Y10 as ON. similar check instructions PT>=, PT<, PT<=, PT=, PT<> can implement the PT pointer pointing judgment check

Example 8

Indirect Addressing of Pointer Type Variables

Pointer type variables can be used in instructions that represent indirect addressing operations on soft components or array variables pointed to by the pointer variable after the address has been obtained through the address operation instruction



When the instruction energy stream is valid, the soft component pointed by the pointer type variable PT0 is added to D100.

If PT0 points to D10, the result of the instruction execution is D300 = D10 + D100。

Description: To use a pointer type variable to indirectly represent the soft component it specifies; a valid pointer address must first be obtained through the pointer address operation instruction.

5. Function Block FB

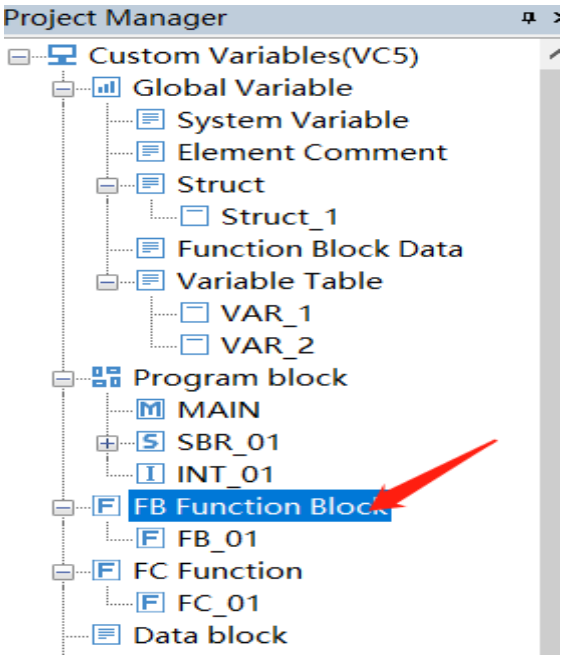
A function block (FB) can abstractly encapsulate reused parts of a program into a common block that can be called repeatedly in the program. The use of encapsulated function blocks in programming increases the efficiency of program development, reduces programming errors, and improves program quality. Function blocks are capable of generating one or more values during execution. Function blocks retain their own special internal variables, and the controller execution system allocates memory to the internal state variables of the function block, which constitute their own state characteristics. For the same parameter input variable value, which may exist in different internal state variables, different calculation results will be obtained.

The basic steps of using function blocks are: New function block -> Function block programming -> Function block instantiation -> Run function block -> Package export function block -> Import function block.

5.1 New Function Block (FB)

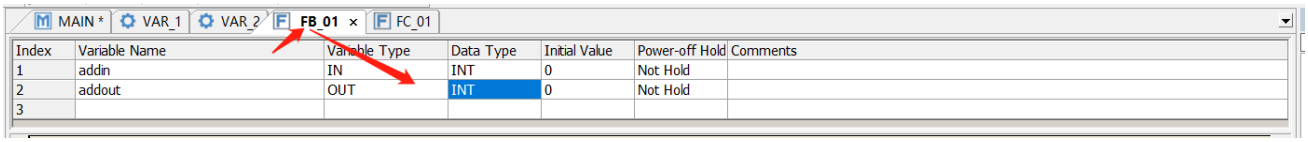
With Auto Studio software, you can create new function blocks

Right-click "FB Function Block" under the "Program Block" node and select New to complete the function block creation. The function block name can be modified by the function block properties.

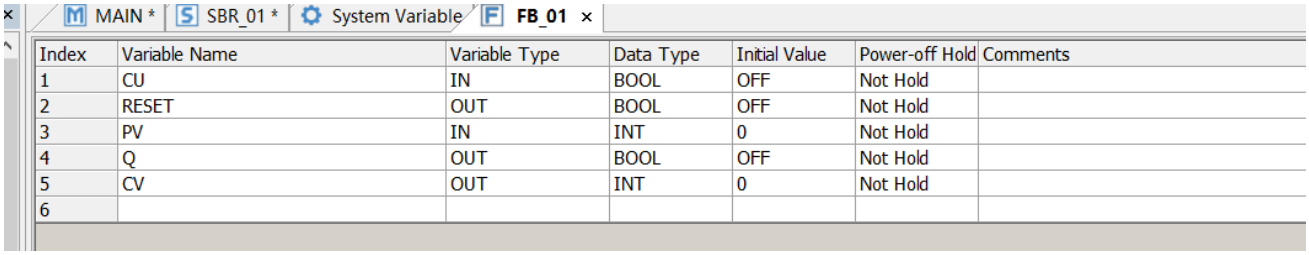


5.2 Function block programming

Double-click the newly created function block under the "Function Block (FB)" node to enter the function block program editing interface. Compared with the normal program editing interface, the function block program editing interface has an additional input/output and local variable definition window.



1. Variable name: The name of the variable



2. Variable type: attribute of function block variable

Variable Type	Type Description	Description
IN	Input Variables	Parameters are provided by the logical block that calls it, and input is passed to the logical block's instructions
OUT	Output Variables	Provide parameters to the logical block that calls it, i.e. output structural data from the logical block
IN_OUT	Input and output variables	Input and output variables can not only be passed into the called logic block, but can also be modified inside the called logic block
TEMP	Local variables	Valid only in this logical block and cannot be accessed externally.

3．Data Type

Variable data types support BOOL, INT, DINT and REAL, and array variables and structures can be defined. If you use structure variables, you need to create structure members in the structure of global variables

4．Initial Value

Set the initial data at the beginning of variable execution.

5．Power down hold

The power-down hold property can set the variable as a hold or non-hold property

● "Non-hold", the variable reverts to the value set in the initial value after power-up；

● If "Hold" is checked in the system parameters, such as "Initialize power-down hold variables when downloading", the variables will be restored to the values set in the initial values when the program is downloaded, otherwise the last run value will be maintained.

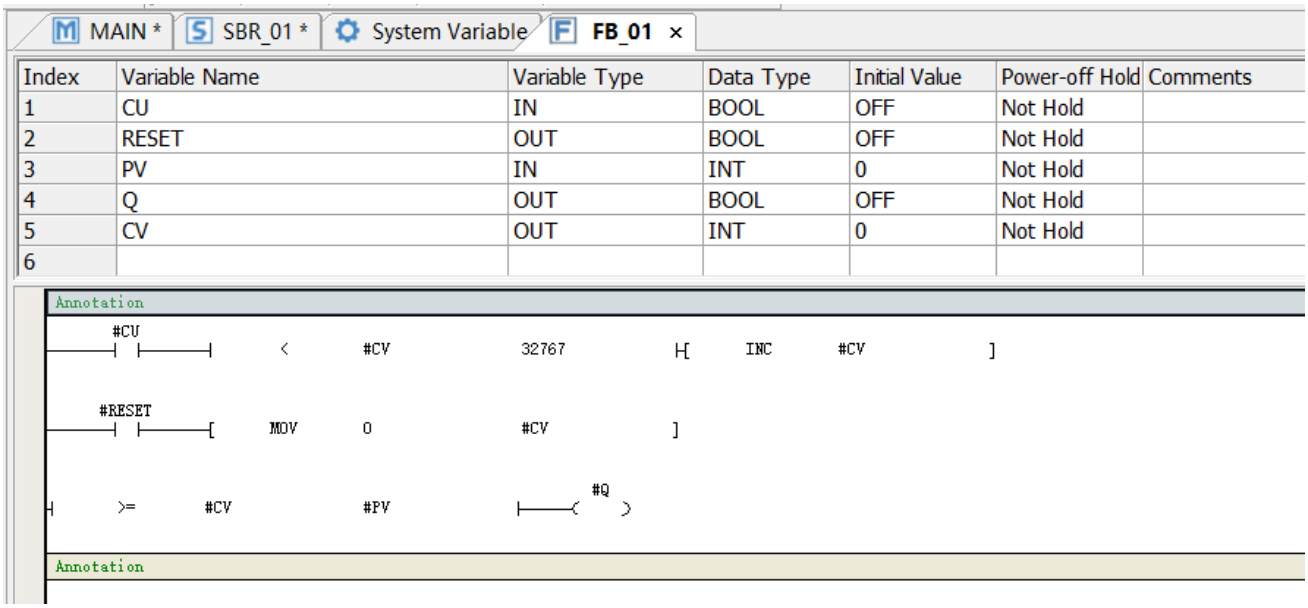
Function block programs are programmed using ladder diagrams. Inside the function block program, functions (FC) or function blocks (FB) can be called, supporting up to 6 levels of nested calls, the

MAIN -> FB_01 -> FB_02 -> FB_03 -> FB_04 -> FB_05 -> FB_06.

Function block programs can use soft components supported by VC5 as global variables in addition to variables, such as SM0

5.2.1 Example - wrapping incremental count with FB

1. The FB program is programmed as follows:

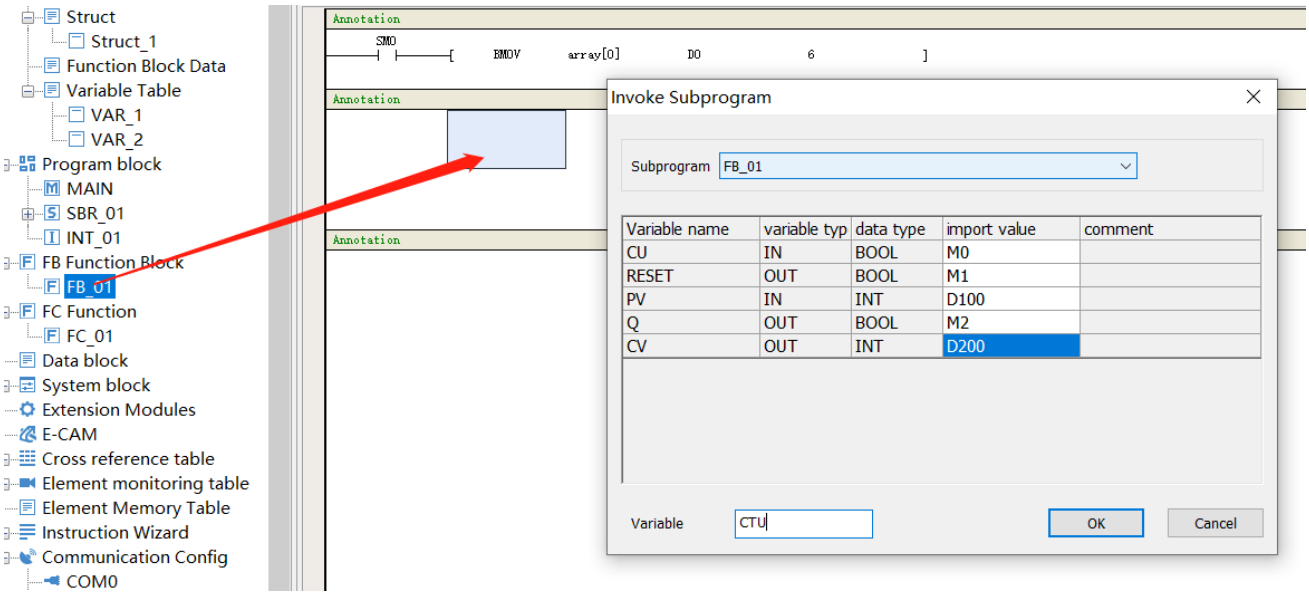


Index	Variable Name	Variable Type	Data Type	Initial Value	Power-off Hold	Comments
1	CU	IN	BOOL	OFF	Not Hold	
2	RESET	OUT	BOOL	OFF	Not Hold	
3	PV	IN	INT	0	Not Hold	
4	Q	OUT	BOOL	OFF	Not Hold	
5	CV	OUT	INT	0	Not Hold	
6						

2. Function block instantiation calls

After writing the FB program and using it in the application, you need to instantiate the function block to call

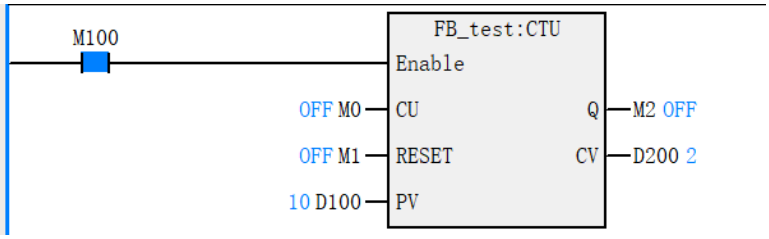
Edit instruction parameters and assign variable names as required by the program to complete the instantiation of function block calls.



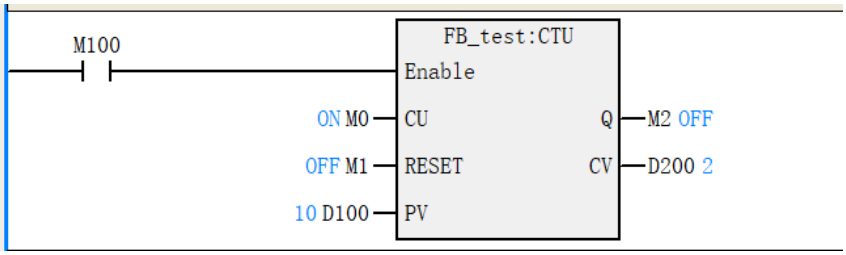
3. Run function blocks

When the function block is instantiated, the En of the function block is connected to the ladder network. when the En network energy flow is active (ON), the function block program is executed and the output of the function block is refreshed and changed according to the input condition and the internal variable state. when the En network energy flow is active (OFF), the function block program is not executed and the function block output is not refreshed.

The counter function block CUT energy flow condition is ON, the function block is executed, and the output CV is added 1 when the input condition CU rising edge changes.



Counter function block CUT energy flow condition is OFF, the function block is not executed, and the output CV is not refreshed when the input condition CU rising edge changes

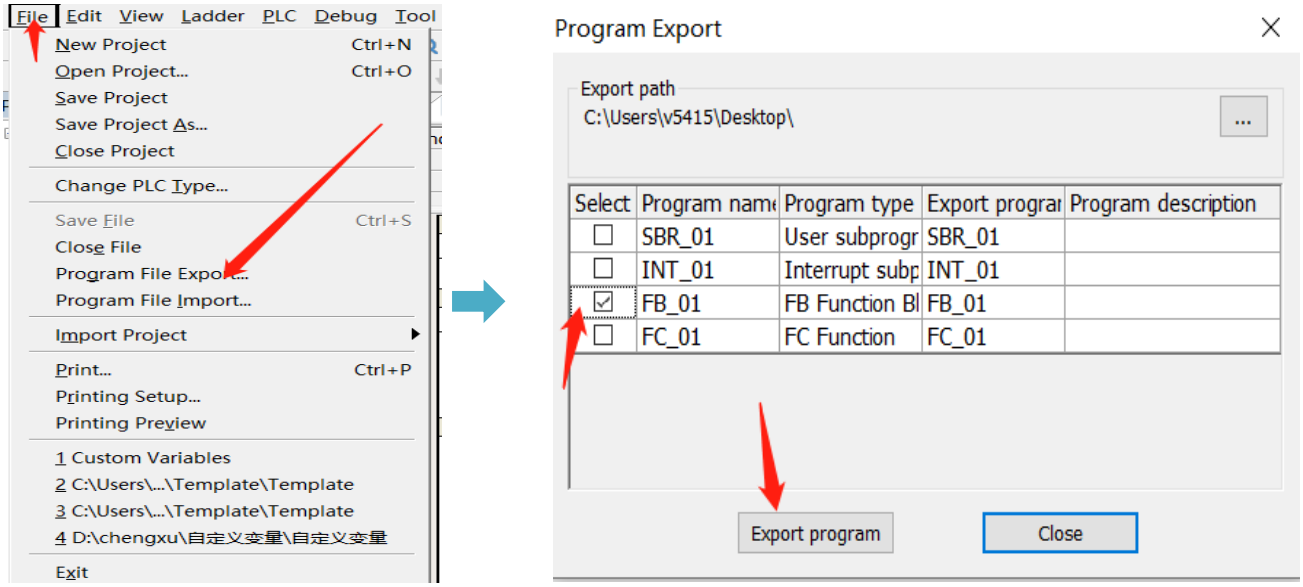


5.2.2 Function block import and export

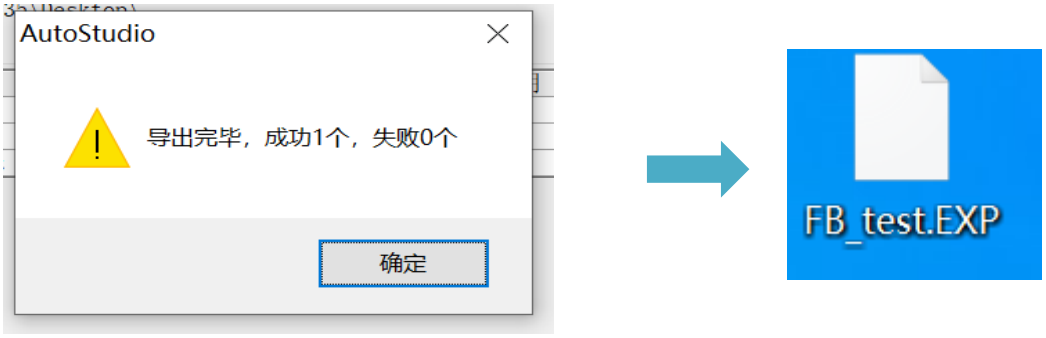
Function block export

The edited and debugged function blocks can be wrapped into files. The function blocks encapsulated into files can be reused in different programs through Auto Studio's file management.

Under File Options, select "Program File Export", select the FB function block to be exported, set the export path, and click Export Program.



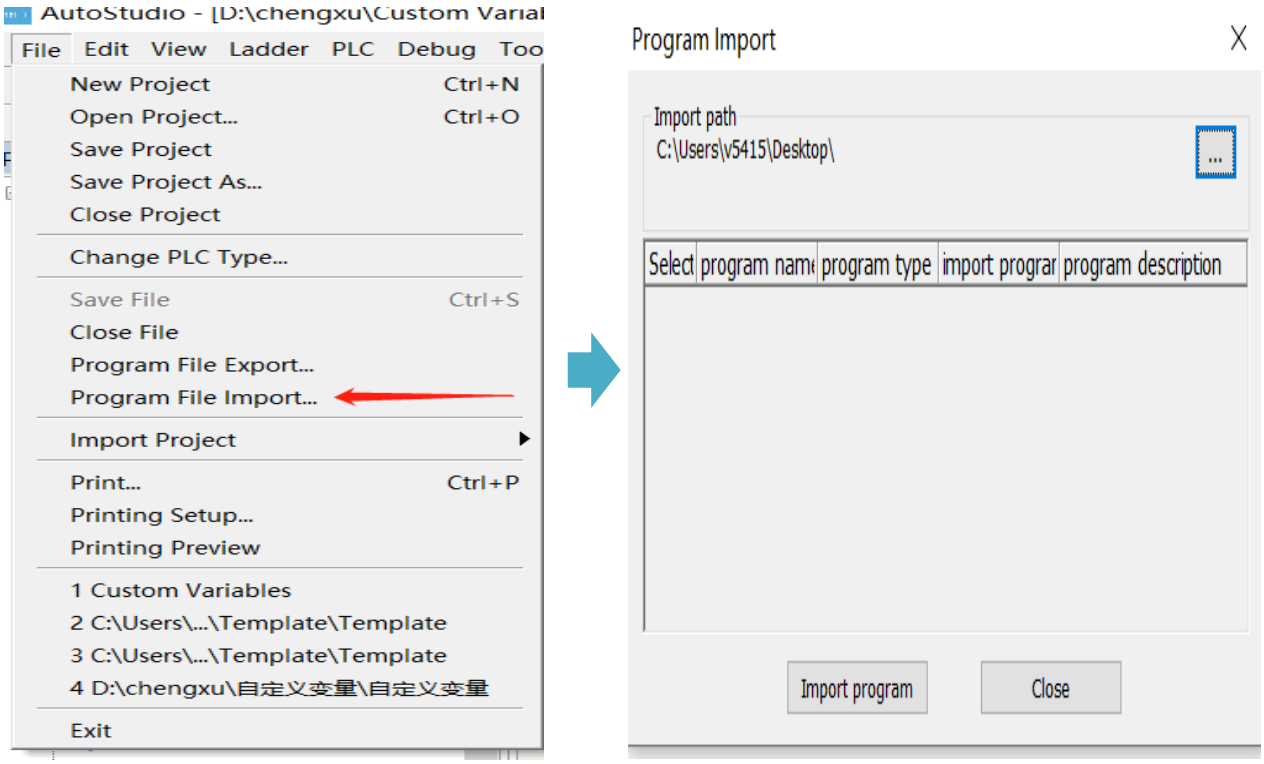
After the successful import of function blocks, the corresponding FB function block program file will be generated under the path



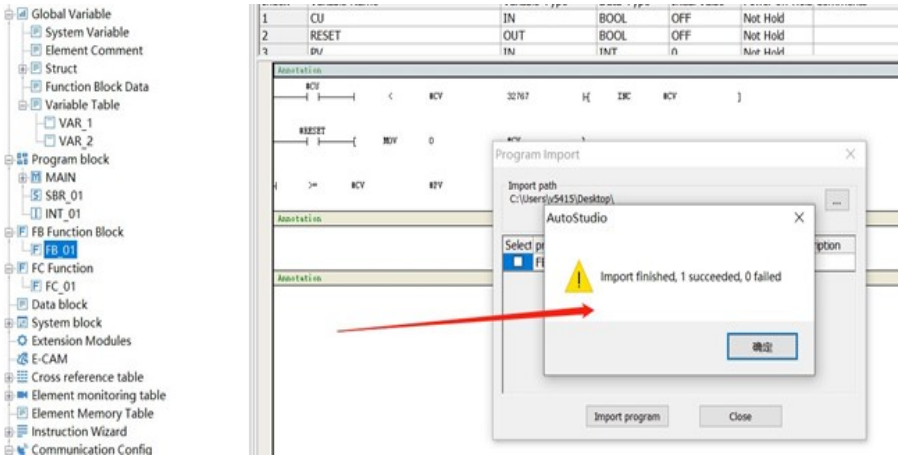
Function block import

Function blocks exported as libraries can be called in other programs by means of import

Under File Options, select "Import Program Files", select the file storage path, select the FB function block to be imported, and click Import Program.



After the successful import, you can see the successfully imported function blocks under FB function blocks.

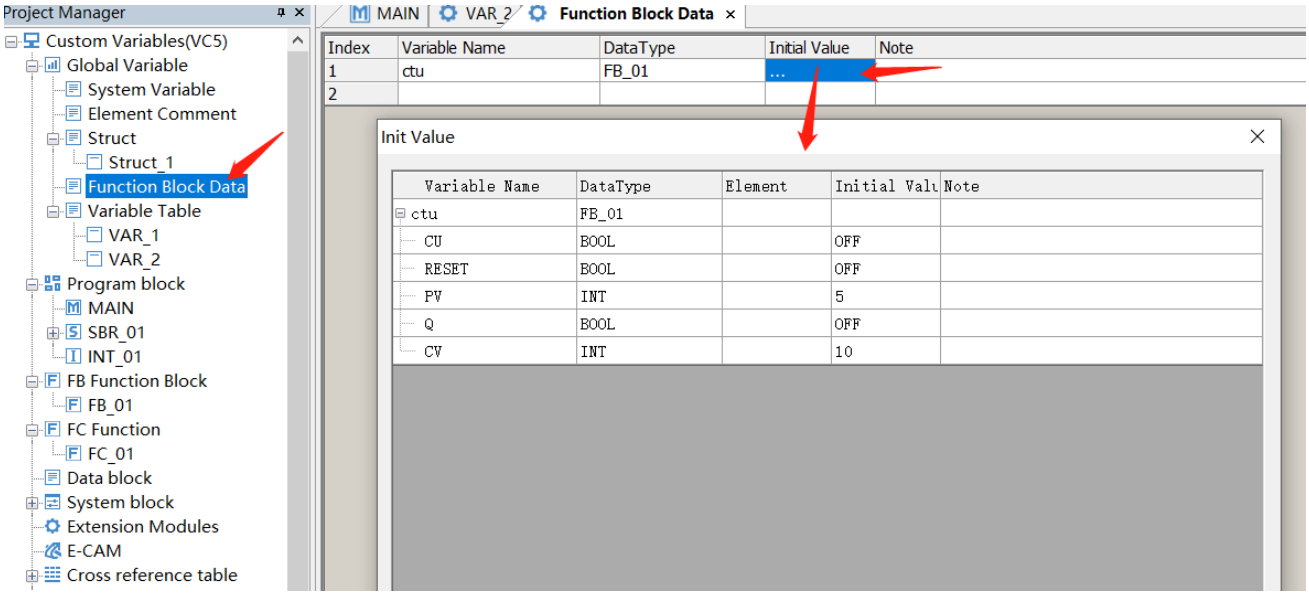


5.2.3 Function block setting initial value

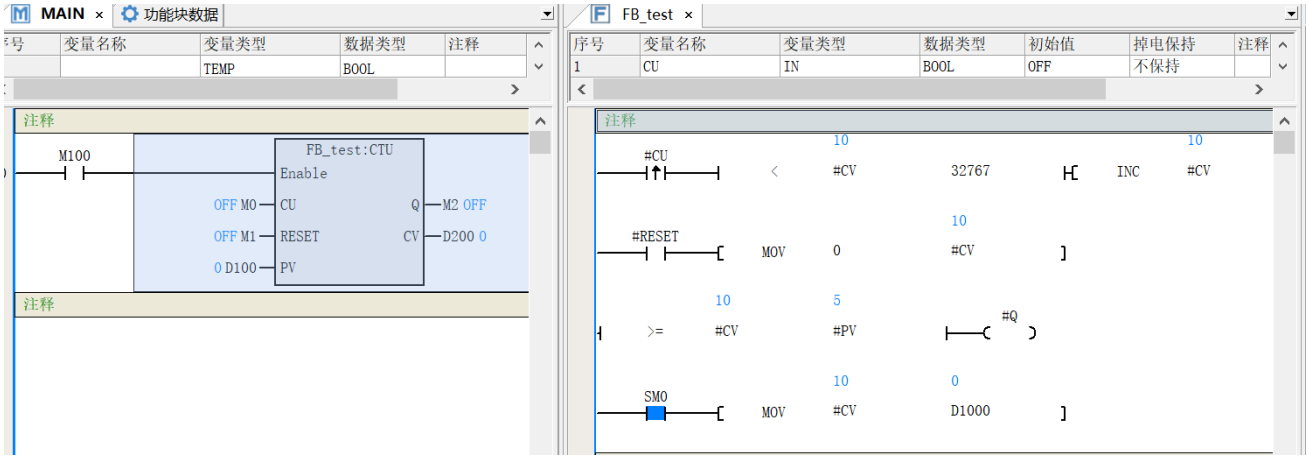
The initial value of FB setting can be modified by either FB type or FB instantiator.

- If the initial value is modified by the FB type, it is equivalent to modifying the initial value of the type.
- If the initial value is modified by the FB instance, it is equivalent to modifying the initial value of the instance.
- If the initial value of the instance is modified, the FB instance member variable will show the modified value.
- If the initial value of the instance is not modified, the FB instance member variable displays the default value.

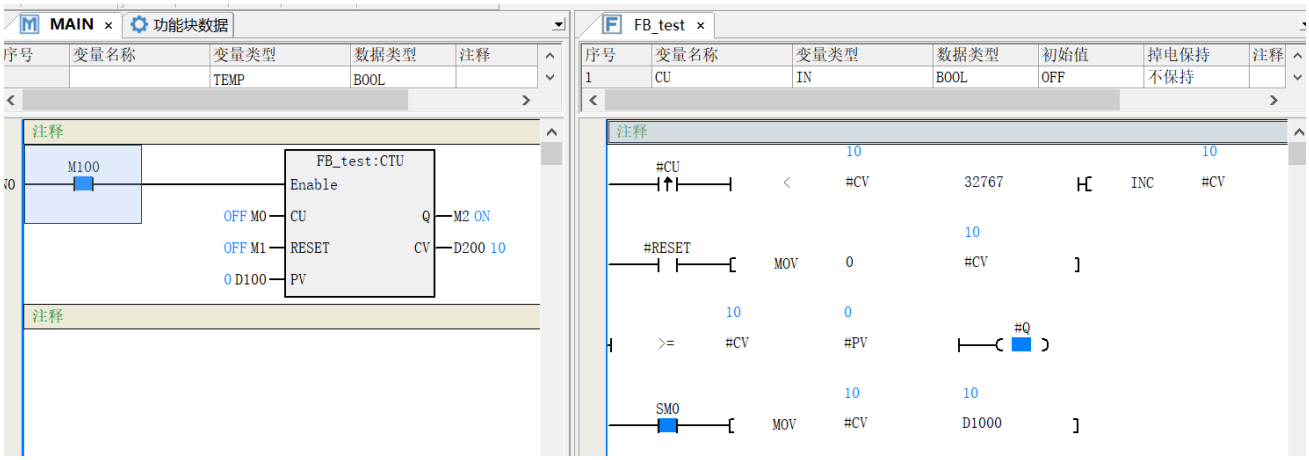
FB modify the initial value



FB function block is not enabled, open the function block, you can see the initial value of the function block



When the FB function block is enabled, the FB instance member variable will display the modified value



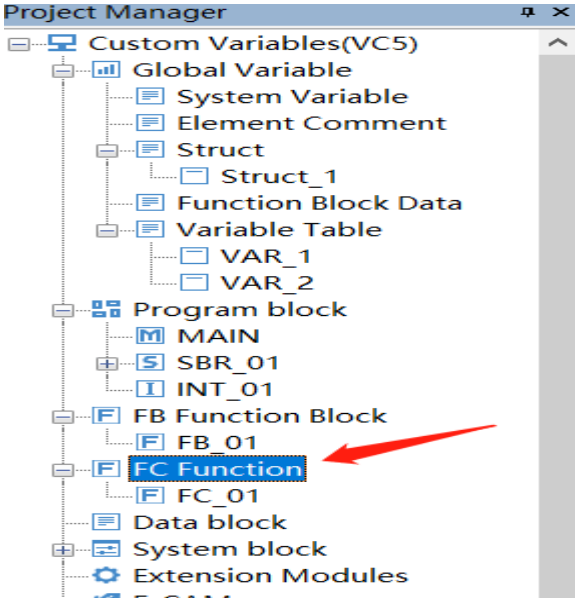
6. Function Block FC

Functions (FC) are independently encapsulated blocks that can define input/output type parameters and can define non-static internal variables, i.e., the same output is obtained when a function is called with the same input parameters. The important feature of a function is that its internal variables are static, there is no internal state storage, and the same input parameters can get the same output, which is the main difference between a function (FC) and a function block (FB). Function (FC), as a basic algorithmic unit, is commonly used in various mathematical operation functions, such as sin(x), sqrt(x), etc. is a typical function type.

The basic steps for using functions are: New Function -> Function Programming -> Call Function -> Run Function -> Wrap Export Function.

6.1 New function

Right-click on "FC Function" under the "Program Block" node and select New to complete the function block creation. The function block name can be modified by the function block properties



6.2 Function Block Programming

Double-click the new function under the "Function (FC)" node to enter the function editor interface. The function program editor interface is similar to the function block, but compared with the normal program editor, there is an additional input/output and local variable definition window

MAIN | SBR_01 | System Variable | FB_01 | FC_01 ×

Index	Variable Name	Variable Type	Data Type	Comments
1	add1	IN	REAL	
2	add2	IN	REAL	
3	smout	OUT	REAL	
4				

In the Input-Output and Local Variable Definition window, you can define input (IN), output (OUT), input-output (INTOUT) and local variables (VAR) for function blocks. Variable data types support BOOL, INT, DINT and REAL, and array variables and structures can be defined. If a structure variable is used, the Structure members need to be created in the structures of global variables.

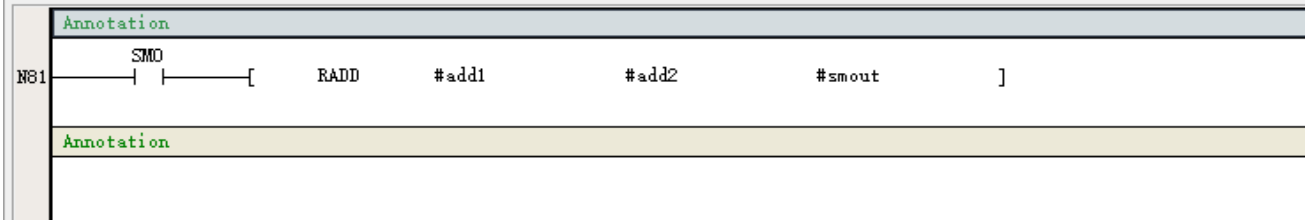
- In contrast to the variables of function blocks, function variables cannot define initial values and all local variables are non-holding.
- Function programs are programmed using ladder diagrams, and inside function programs, functions (FC) can be called. The function itself can be called by other functions, function blocks, and programs.
- Function programs can use SM0 as a constant ON variable in addition to variables.
- Instructions related to state or multi-cycle execution, such as LDP, MC_Power, etc., cannot be used in function programs.

6.2.1 Example - Wrapping addition functions with FC

1. The FC functions are programmed as follows:

MAIN | SBR_01 | System Variable | FB_01 | FC_01 ×

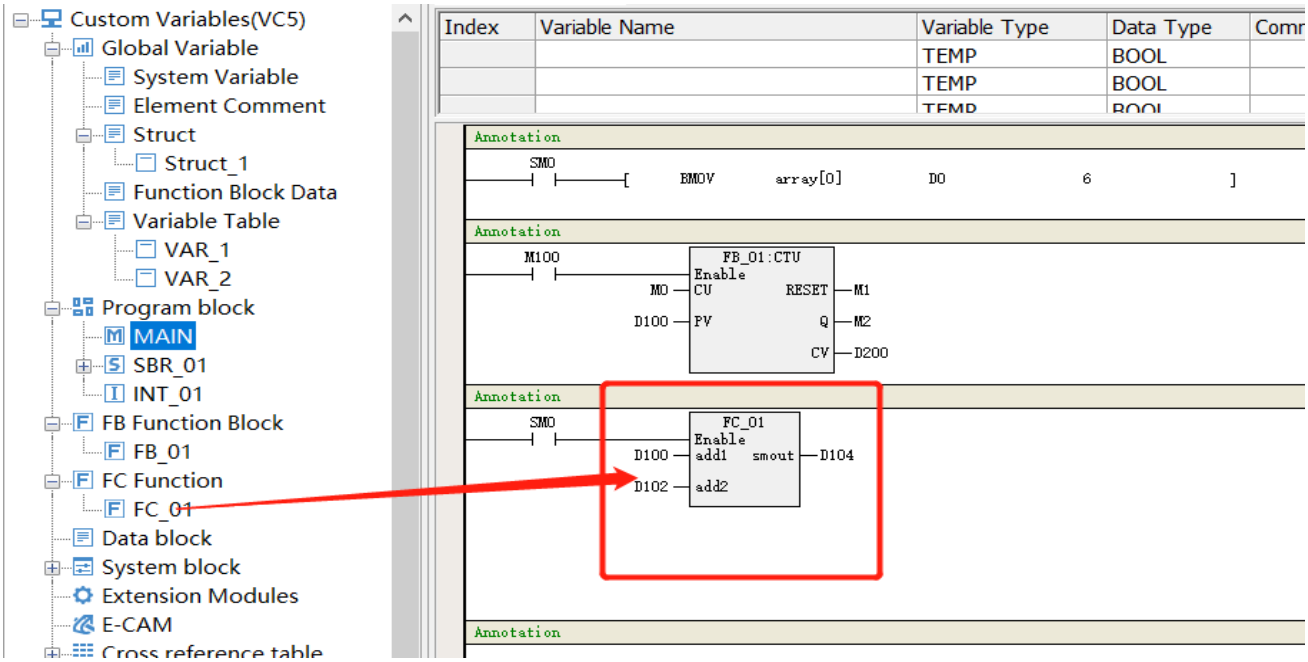
Index	Variable Name	Variable Type	Data Type	Comments
1	add1	IN	REAL	
2	add2	IN	REAL	
3	smout	OUT	REAL	
4				



2. FC function calls

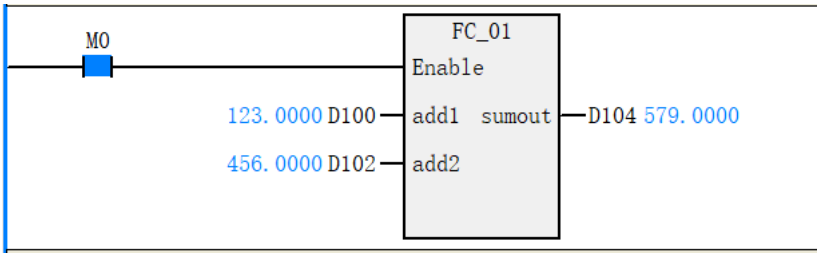
After writing the FC program, you can use it in your application or call it directly to use it.

Edit the instruction parameters and assign variable names as required by the program to complete the instantiation call of the function block.



3. Run the function

When the En network is active (ON), the function program is executed and the output of the function is refreshed according to the input state operation; when the En network is active (OFF), the function program is not executed and the output of the function block is not refreshed.



6.2.2 Function block import and export

The procedure for wrapping functions is similar to that for function blocks, please refer to the section "Import and Export of Function Blocks".